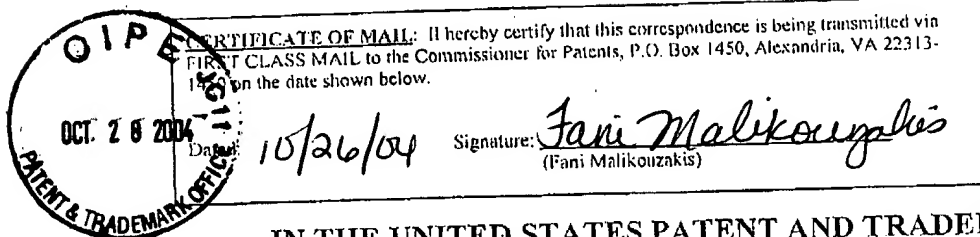


THEOR 205.1 US (10107436)



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:
Charles PACLAT et al.

Application No.: 09/975,945

Group Art Unit: 2124

Filed: October 11, 2001

Examiner: Anil KHATRI

For: METHOD FOR DEVELOPING BUSINESS
COMPONENTS

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION UNDER 37 C.F.R. § 131

Dear Sir:

I, CHARLES PACLAT DECLARE AND STATE THAT:

1. I am the inventor of the "Method for Developing Business Components" described and claimed in the above-identified U.S. Patent Application ("the Present Application").
2. I was advised and therefore believe that the Present Application claims the benefit of U.S. Provisional Patent Application Serial No. 60/239,409 filed October 11, 2000 ("the Provisional Application").
3. The Present Application was assigned to BEA Systems, Inc. by me on December 12, 2001. Exhibit A is a copy of the Notice of Recordation of Assignment document evidencing the recordation of the assignment on January 22, 2002.

THEOR 205.1 US (10107436)

4. Well prior to August 24, 2000, the claimed invention shown and described in the provisional and present applications was conceived and reduced to practice.

5. Exhibit B is a document written well prior to August 24, 2000 and provides an overview of New Component Development Process. (The actual data has been redacted from this document).

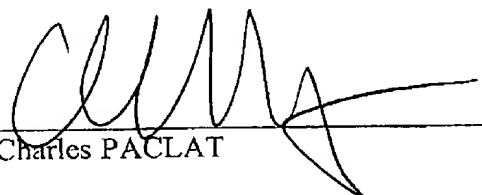
6. Exhibit C is a document written well prior to August 24, 2000 and provides a description of the New Component Development Process. (The actual data has been redacted from this document).

7. We further declare that all statement made herein of our own knowledge are true and that all statements made on information and belief are believed to be true; and that these statements were made with the knowledge that willful, false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful, false statements may jeopardize the validity of the application or any patent issuing thereon.

Date:

10/21/2004

Signed:


Charles PACLAT



MARCH 20, 2002

PTAS

FULBRIGHT & JAWORSKI L.L.P.
C. ANDREW IM
666 FIFTH AVE.
NEW YORK, NY 10103

Chief Information Officer
Washington, DC 20231
www.uspto.gov



101960028A

UNITED STATES PATENT AND TRADEMARK OFFICE
NOTICE OF RECORDATION OF ASSIGNMENT DOCUMENT

THE ENCLOSED DOCUMENT HAS BEEN RECORDED BY THE ASSIGNMENT DIVISION OF THE U.S. PATENT AND TRADEMARK OFFICE. A COMPLETE MICROFILM COPY IS AVAILABLE AT THE ASSIGNMENT SEARCH ROOM ON THE REEL AND FRAME NUMBER REFERENCED BELOW.

PLEASE REVIEW ALL INFORMATION CONTAINED ON THIS NOTICE. THE INFORMATION CONTAINED ON THIS RECORDATION NOTICE REFLECTS THE DATA PRESENT IN THE PATENT AND TRADEMARK ASSIGNMENT SYSTEM. IF YOU SHOULD FIND ANY ERRORS OR HAVE QUESTIONS CONCERNING THIS NOTICE, YOU MAY CONTACT THE EMPLOYEE WHOSE NAME APPEARS ON THIS NOTICE AT 703-308-9723. PLEASE SEND REQUEST FOR CORRECTION TO: U.S. PATENT AND TRADEMARK OFFICE, ASSIGNMENT DIVISION, BOX ASSIGNMENTS, CG-4, 1213 JEFFERSON DAVIS HWY, SUITE 320, WASHINGTON, D.C. 20231.

RECORDATION DATE: 01/22/2002

REEL/FRAME: 012497/0313
NUMBER OF PAGES: 3

BRIEF: ASSIGNMENT OF ASSIGNOR'S INTEREST (SEE DOCUMENT FOR DETAILS).

ASSIGNOR:

PACLAT, CHARLES

DOC DATE: 12/12/2001

ASSIGNEE:

BEA SYSTEMS, INC.
2315 NORTH FIRST STREET
SAN JOSE, CALIFORNIA 95134

SERIAL NUMBER: 09975945
PATENT NUMBER:

FILING DATE: 10/11/2001
ISSUE DATE:

KIMBERLY WHITE, EXAMINER
ASSIGNMENT DIVISION
OFFICE OF PUBLIC RECORDS

FULBRIGHT & JAWORSKI, LLP
NEW YORK DOCKETING

Docketed ☐ Not Required ☐
Previously ☐ Updated ☒

Docket No: Theor 205-1 CPT

Action: _____

Reminder: _____

Date: Due/Done _____

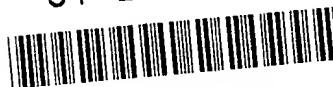
Initials: WJ

FULBRIGHT & JAWORSKI, LLP
NEW YORK OFFICE

2002 MAR 26 A 10:51

DOCKET DEPT.
RECEIVED

01-28-2002



101960028

ET

U.S. Department of Commerce

Patent and Trademark Office

THEOR 205.1 (10107436)

To the Honorable Commissioner of Patents and Trademarks: Please record the attached original documents or copy thereof.

1. Name of conveying party(ies):

Charles PACLAT

Additional name(s) of conveying party(ies) attached? ☐ Yes ☒ No

3. Nature of conveyance:

- ☒ Assignment ☐ Merger
- ☐ Security Agreement ☐ Change of Name
- ☐ Other _____

Execution Date: December 12, 2001

2. Name and address of receiving party(ies):

Name: BEA Systems, Inc.

Internal Address: _____

Street Address: 2315 North First StreetSan Jose, California 95134Additional name(s) & address(es) attached? ☐ Yes ☒ No

4. Application number(s) or patent number(s):

If this document is being filed together with a new application, the execution date of the application is: _____

A. Patent Application No.(s)

B. Patent No.(s)

09/975,945Additional numbers attached? ☐ Yes ☒ No

5. Name and address of party to whom correspondence concerning document should be mailed:

Name: C. Andrew ImInternal Address: FULBRIGHT & JAWORSKI L.L.P.Street Address: 666 Fifth AvenueCity: New York State: NY Zip: 101036. Total number of applications and patents involved: ☒7. Total fee (37 C.F.R. § 3.41) \$40.00☒ Enclosed☐ Authorization to be charged to deposit account8. Deposit Account Number: 500624

(Attach duplicate of this page if paying by Deposit Account)

DO NOT USE THIS SPACE

9. Statement and signature.

To the best of my knowledge and belief, the foregoing information is true and correct and any attached copy is a true copy of the original document.

C. Andrew Im

Name of Person Signing

Signature

December 13, 2001

Date

Total number of pages including cover sheet, attachment, and document: ☐

01/25/2002 LMJELLER 00000177 09975945 Mail documents to be recorded with required cover sheet information to:

01 FC:581

40.00 OP

Commissioner of Patent and Trademarks, Box Assignments
Washington, D.C. 20231

Docket Number: THEOR 205.1 US - CAI

ASSIGNMENT

WHEREAS, I, as a below named inventor, residing at the address stated next to my name, am a sole inventor (if only one name is listed below) or a joint inventor (if plural names are listed below) of certain new and useful improvements in METHOD FOR DEVELOPING BUSINESS COMPONENTS for which an application for Letters Patent of the United States of America was executed by me on the date indicated next to my name and address;

AND WHEREAS, BEA Systems, Inc. with principal offices located 2315 North First Street, San Jose, CA 95134 (hereinafter referenced as ASSIGNEE) is desirous of acquiring all interest in, to and under said invention, said application disclosing the invention and in, to and under any Letters Patent or similar legal protection which may be granted therefor in the United States and in any and all foreign countries;

NOW THEREFORE, in consideration of the sum of One Dollar (\$1.00), and other good and valuable consideration, the receipt and sufficiency of which are hereby acknowledged, I, as a sole or joint inventor as indicated below, by these presents do hereby assign, sell and transfer unto the said ASSIGNEE, its successors, assigns, and legal representatives, the entire right, title and interest in the said invention, said application, including any divisions and continuations thereof, and in and to any and all Letters Patent of the United States, and countries foreign thereto, which may be granted for said invention, and in and to any and all priority rights and/or convention rights under the International Convention for the Protection of Industrial Property, Inter-American Convention Relating to Patents, Designs and Industrial Models, and any other international agreements to which the United States of America adheres, and to any other benefits accruing or to accrue to me with respect to the filing of applications for patents or securing of patents in the United States and countries foreign thereto, and I hereby authorize and request the Commissioner of Patents to issue the said United States Letters Patent to said ASSIGNEE, as the assignee of the whole right, title and interest thereto;

And I further agree to execute all necessary or desirable and lawful future documents, including assignments in favor of ASSIGNEE or its designee, as ASSIGNEE or its successors, assigns and legal representatives may from time-to-time present to me and without further remuneration, in order to perfect title in said invention, modifications, and improvements in said invention, applications and Letters Patent of the United States and countries foreign thereto;

And I further agree to properly execute and deliver and without further remuneration, such necessary or desirable and lawful papers for application for foreign patents, for filing subdivisions of said application for patent, and or, for obtaining any reissue or reissues of any Letters patent which may be granted for my aforesaid invention, as the ASSIGNEE thereof shall hereafter require and prepare at its own expense;

Docket Number: THEOR 205.1 US - CAI

And I further agree that ASSIGNEE will, upon its request, be provided promptly with all pertinent facts and documents relating to said application, said invention and said Letters Patent and legal equivalents in foreign countries as may be known and accessible to me and will testify as to the same in any interference or litigation related thereto;

And I hereby covenant that no assignment, sale, agreement or encumbrance has been or will be made or entered into which would conflict with this assignment and sale.

This assignment executed on the dates indicated below.

Charles PACLAT

Name of first or sole inventor

114 Wolcott Street
Medford, MA 02155

Residence of first or sole inventor

Signature of first or sole inventor

12/12/2001
Date of this assignment



New Component Development - Overview



Agenda



- New Component Development Process
- Existing eBusiness Components
- Components in Development
- Next Generation Components

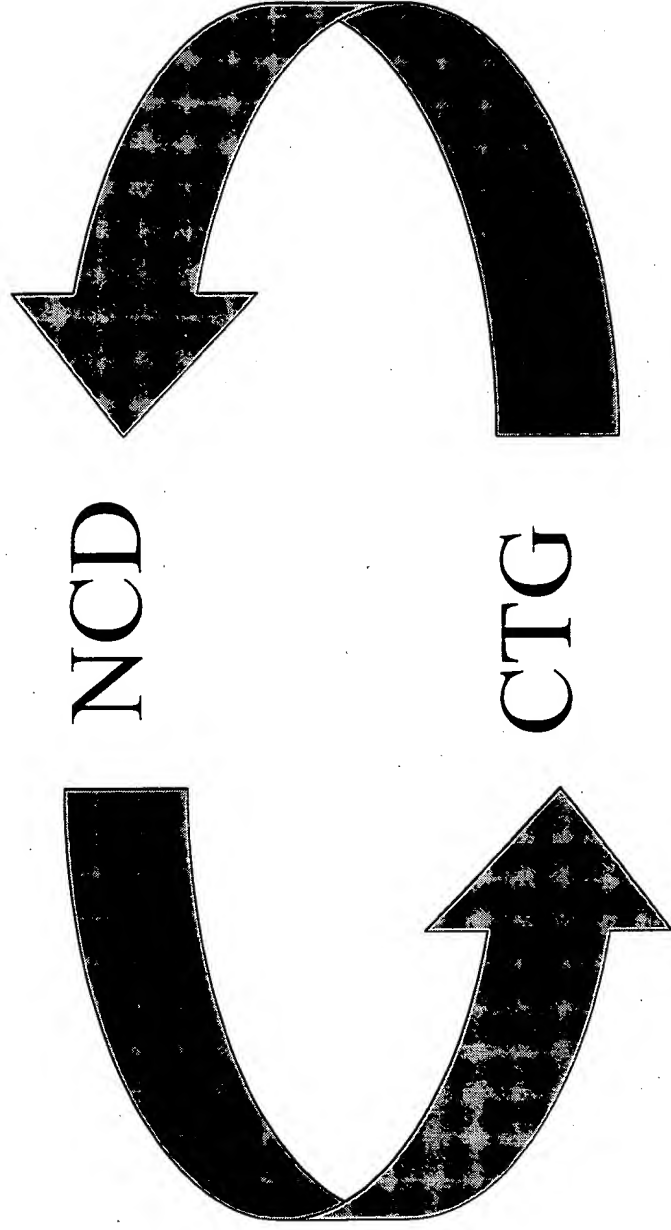
New Component Development Process

.....



- Developing Business Components is different...
 - from developing end-user applications
 - from developing software tools
- Greater focus on OO Analysis and Design...
 - Business domain knowledge is very important
 - Components must be reusable
 - Components must be extensible
- Component development tools...
 - Modeling and code generation tools
 - Allow customers to adapt the components

NCD and Core Technologies



**Better tools mean better components.
More components result in better tools.**

New Component Development Process - Motivation



- Learn from the process of building the components for eBSC
- Divide the process into manageable steps
- Facilitate communication within the organization
- Put in place practices which will enable the process to scale

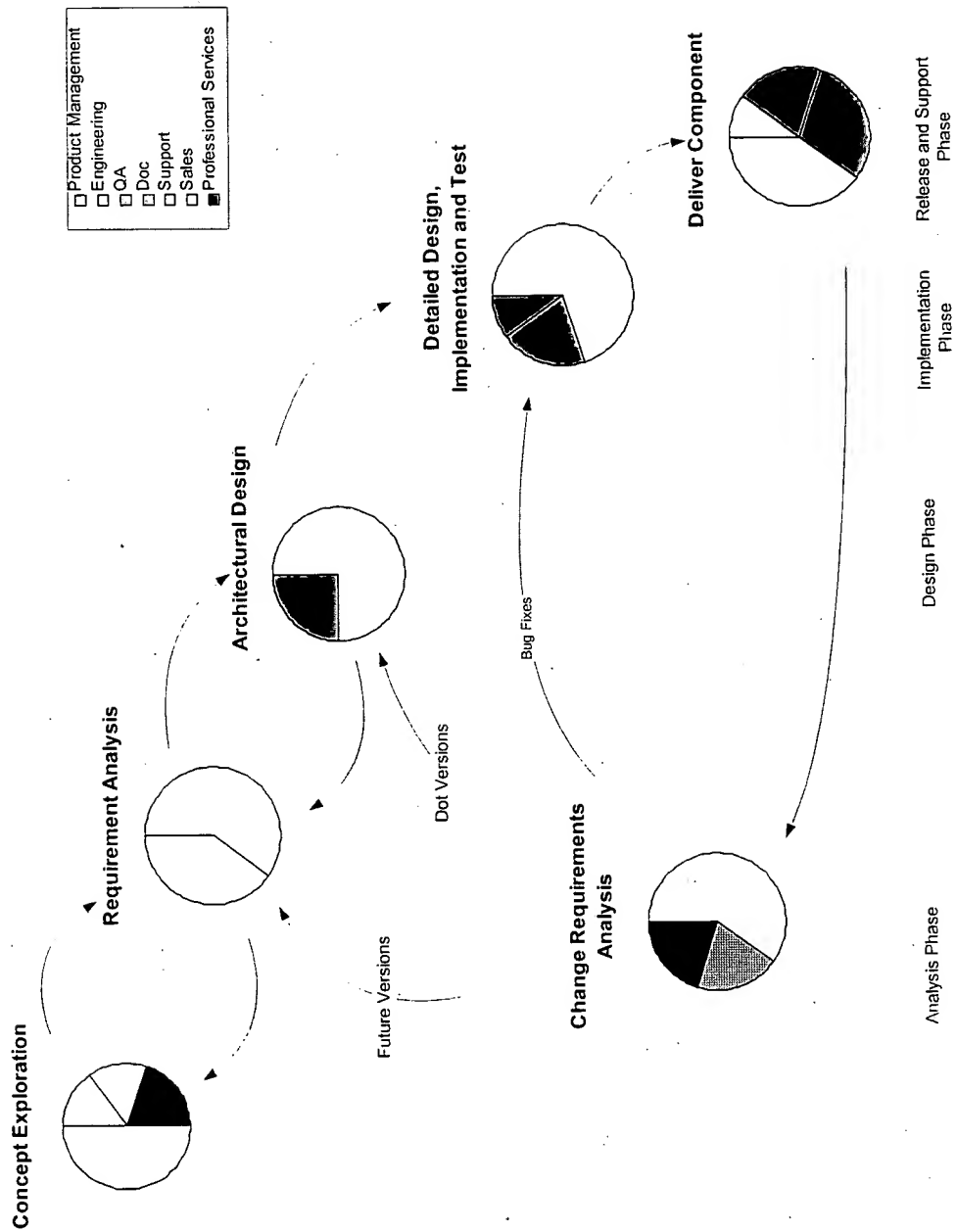
New Component Development Process - Philosophies



- **Adaptable** - Just enough process without being burdensome
- **Time to Market** - Analysis is thorough but has specific deliverables so as not to drag out
- **Divide and Conquer** – Recursive analysis
- **Iterative** – Repeated short timeframe deliverables
- **Efficient** - There are no hand offs
- **Parallel** – NCD and Phases can occur in parallel

“Productivity should be limited only by our ability to add talented new members to the team.”

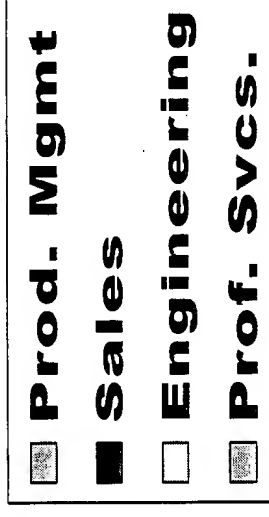
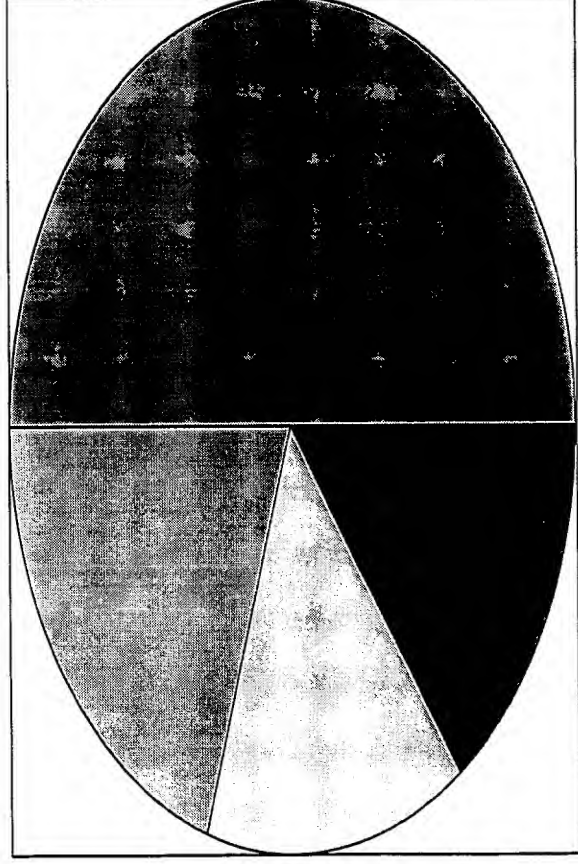
NCD from start to finish



Concept Exploration



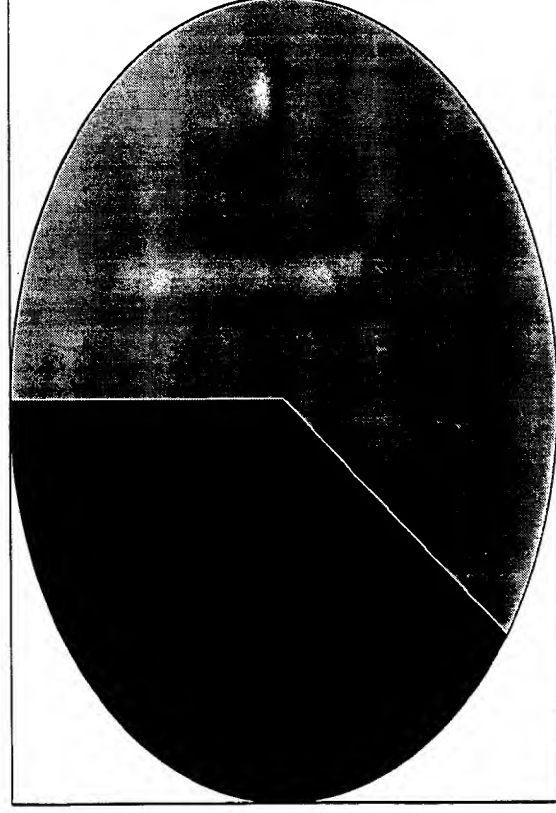
- Decide which component domains should be pursued and provide overall direction.
- Result is a recommendation for an NCD Analysis



Requirements Analysis



Detailed analysis of the functional requirements from a standpoint of implementing the functionality as a set of components.



 **Prod. Mgmt**
 **Engineering**

Requirements Analysis-Deliverables

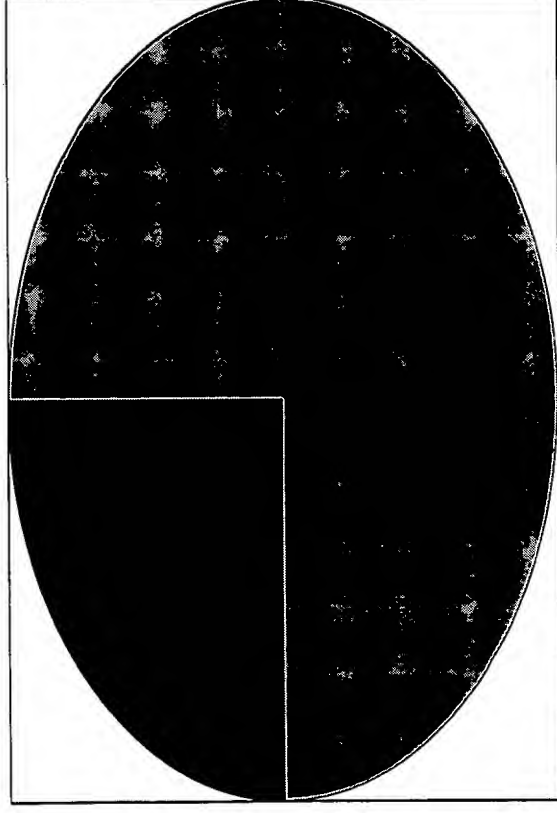


- Summary of Scope
- Collection of Resources
 - Industry Analysts
 - Related Industry Standards
 - Commercial Packages
 - Competitors
 - Partners
 - Related Engagements and System Integrators
 - **In-house Expertise (Hiring Domain Experts)**
- **Functionality Matrix**

Architectural Design



Turning the functional requirements into an object oriented model that encapsulates the data model and the process model



| | |
|---|--------------------|
|  | Engineering |
|  | QA |

Architectural Design – Deliverables

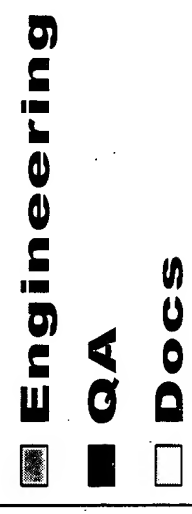
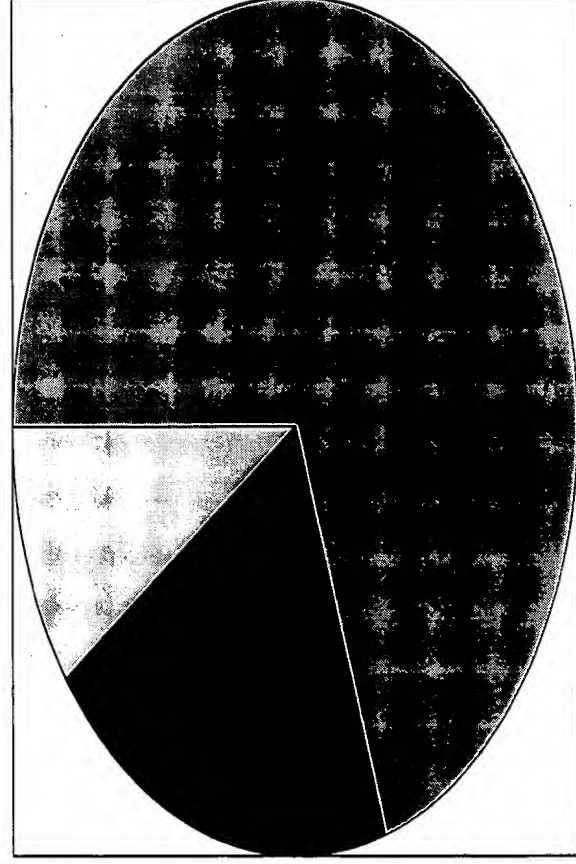


- Use Cases
- UML Models
- Java Doc
- Test Plans
- Generated Classes
- Design Review

Implementation

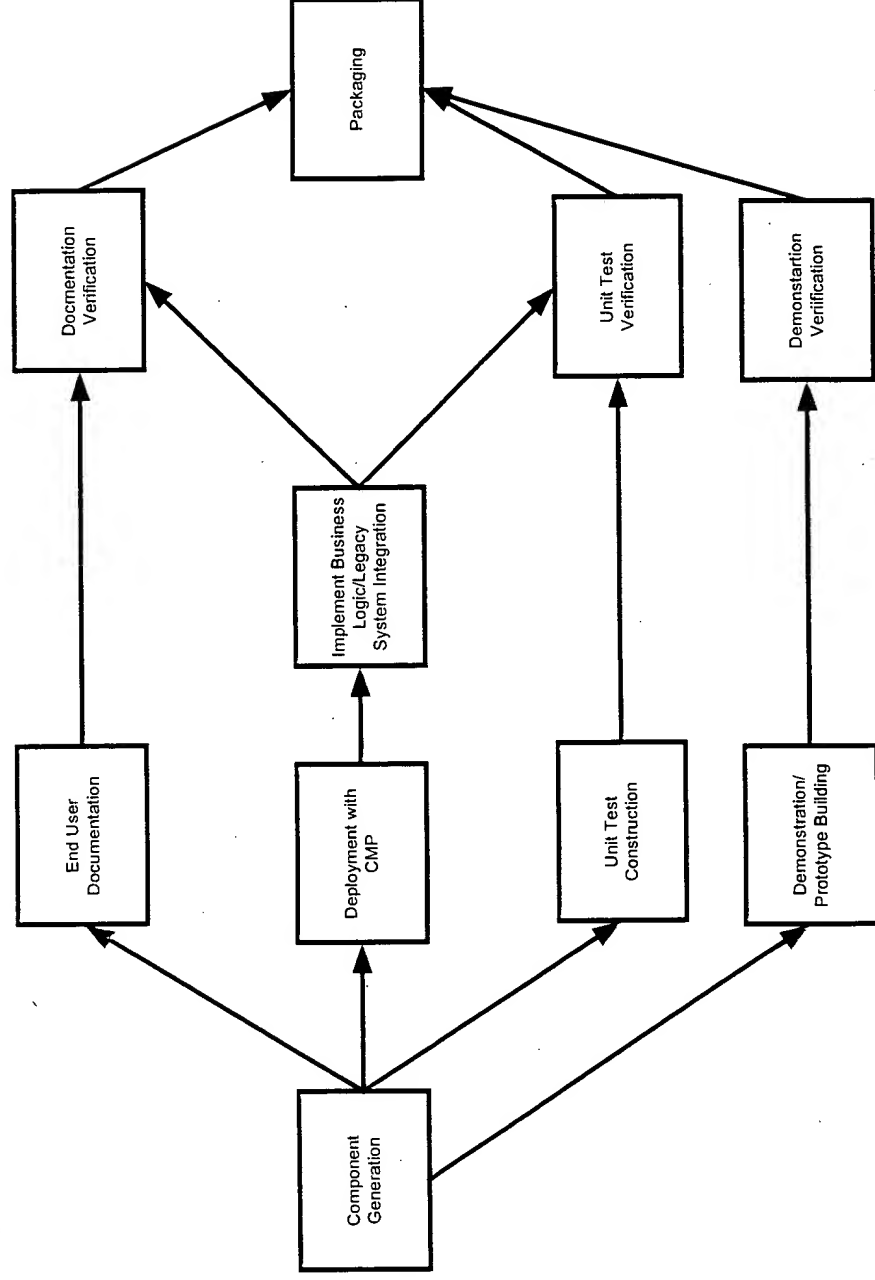


- The implementation phase is structured to maximize parallel development.





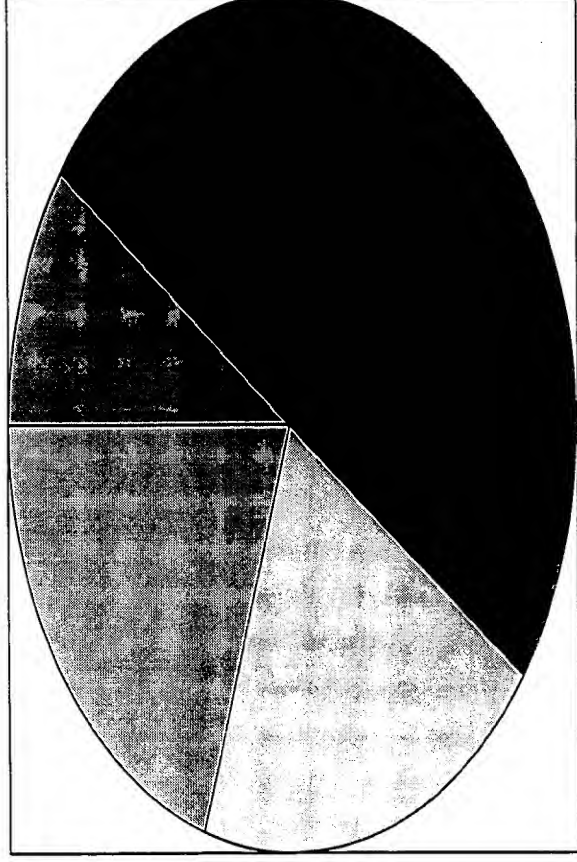
Implementation - Process



Delivering Components



- This is the process of packaging and delivering the components

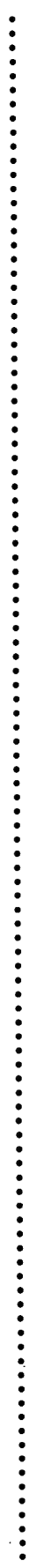




Classifying Components

- Integration – wrapper around a provider
 - Business logic is too complex
 - Access to 3rd Party service providers
 - Provide plug and play integration
- Build – we implement the business logic
 - Opportunity to provide real value
 - Cheaper than if we partner
 - Core functionality where we need flexibility
- Buy – re-factoring or repackaging of existing product
 - Time to market required
 - Cheaper to buy than to build

JumpStart 1.6.1



- Foundation and Smart Generator
 - Part of the Core Technologies
 - Base classes from which Patterns are built
- Axiom
 - Cross vertical components built on foundation
- eBusiness
 - B2C
 - Retail Sales model

JumpStart 1.6.1

Axiom



- Accounting
 - Accounting from “Analysis Patterns”
 - Sales, Inventory, Banking, Portfolio
- Contact
 - Customer contact information
 - Addresses, Phone Numbers, E-mail, URL, Postal Codes, Credit Card
- Message (not JMS)
 - Simple persistent text messages.

JumpStart 1.6.1

Axiom



- Units
 - Support for table driven unit conversion
- Workflow
 - Internal State management that guides the usage of an object.
- Utilities
 - AlphaNumericSequencer

JumpStart 1.6.1

eBusiness – Shopping



- Customer
 - Contact plus credit card
- Item
 - Cataloging and pricing
- Shopping Advisor
 - Search and Recommendation engine
- Order
 - Cart and basic order fulfillment
- Session
 - Anonymous browse and checkout.

JumpStart 1.6.1

eBusiness – Back Office



- Shipping
 - Shipping cost calculation mechanisms
- Invoicing
 - Generic interface to deliver and invoice
- Inventory
 - Generic interface for checking and updating inventory
- Trouble Ticket
 - Basis for Support Desk functionality



Components in Development



- Filling in the Gaps
 - Payment
 - Tax
 - Order
 - Accounting
- Marketing Oriented Stuff
 - Gift Registry

Components in Development - Payment



Challenge: BuyBeans simply recorded the credit card information and assumed that the authorization is done offline.

Solution: Develop a generic set of payment mechanism one of which was Credit Card. Provide integration with first tier authorization products.

Partners:

- CyberCash(80%) – Service based (JNI)
- PayLinx(10%) – Client hosted (Java API)

Issues: Need to support many additional payment instruments has made it more powerful and more difficult to implement

Components in Development – Tax Calculation

.....



Challenge: “Real” eCommerce sites need a solution to collecting sales taxes. Existing packages are not easily integrated into the J2EE environment.

Solution: Design and develop a generic set of session components to front end various tax providers.

Partners:

- Taxware(80%) - JNI
- Quantum(10%) – Beta users of Java interface

Issues: Getting support from the vendors has been difficult. BEA and getting real customers should help this process.

Components in Development- Order Management



Challenge: The order model in 1.6.1 did not support partial shipment, partial invoice, or returns. Driven by requests from site developers that wanted to display order status to their customers.

Solution: Build a model that supports these concepts so that we can integrate with complex order fulfillment vendors and ERP solutions.

Issues: This is an extremely complex domain to model. Pricing engines and incentive management can be very difficult to implement.

Next Generation Components



.....

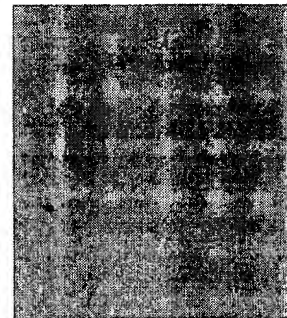
- Accounting Patterns
 - Martin Fowler
 - Sales Accounting
 - Inventory
- eBanking
 - Wingspan/Sanchez
 - OFX based models
- eFinancial
- eMerchandising
 - Gift Registry
 - Sexy
 - No prepackaged

Next Generation Components



.....

- Negotiation
 - Auction
 - Exchange
- Business Process Workflow
 - Order Management
 - Issue Tracking
- Pricing Engine
 - Incentives
 - Discounts
- Payment Mechanisms
- Knowledge Management
- Content Management
 - Catalog
 - Ad Servers
- Enhanced Shipping
 - Cost Calculation
 - Tracking

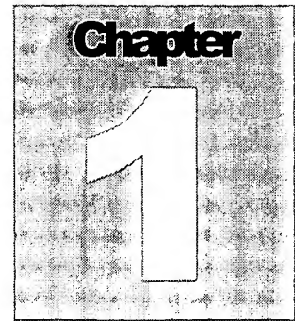


A process for the discovery and implementation of generic business components.

The NOD Process

A PROCESS FOR THE DISCOVERY AND IMPLEMENTATION OF GENERIC BUSINESS
COMPONENTS

The New Component Development Process



Introduction

Building Business Objects

This document outlines a process by which BEA/eSolutions will create object oriented representations of the software components that facilitate a vast array of business solutions. It identifies the key steps that are needed to build these components and provides guidelines for the order of their execution. Underlying the concepts in this document are some key guiding principles that should be kept in mind throughout.

Developing Business Objects

Developing business objects differs from developing end user applications. The traditional end user is the person that is using an application to complete their daily tasks. In the case of business objects the end users are software developers and business analysts. Ultimately it is these users that will be customizing these components to create the complete solutions that the customers demand.

Business objects also differ from the development of traditional software tools and infrastructure. This end user in these cases is only the software developer. Such tools rarely contain business domain specific logic. They can be designed and developed with no input from the business domain experts.

Business object development requires much greater collaboration between the business analysts and the software engineering teams. The result will be well-structured components that capture the most important facets of a business domain.

Time to Market

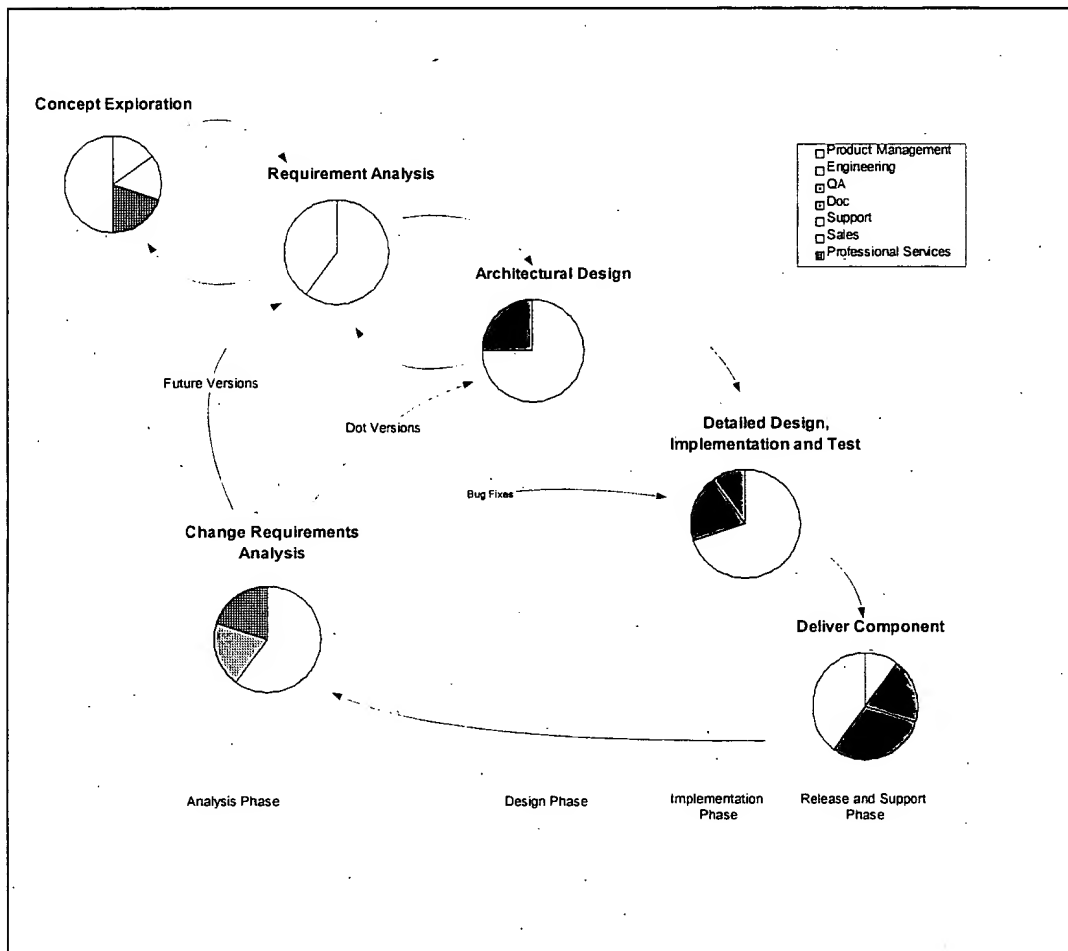
The ability to deliver components in a timely manner is an overriding concern in today's extremely competitive marketplace. One of the key premises is that up front research will allow us to define the segments of the space where we can provide value added and where we should form relationships. With this in mind it is important to be thorough but timely in the analysis. The typical up front analysis should consume no more than a few weeks.

Iterative Development

It should be stressed that these steps are not to be executed in a rigidly defined sequence. The only hard and fast rule is that a thorough market analysis must be done up front. This will allow the design and implementation phases to be completed in an environment of full disclosure. It is expected that a very scaled back version of the components will be delivered after the first iteration. Each of the iterations will provide feed back into the analysis and design, and will result in a product that is increasingly complete.

Flow of Participation

An important technique that is used in the NCD Process is the overlapping of team members as development passes through the various phases. Each phase of development will involve participants from the functional organizations that follow it in the process.



Concept Exploration

The initiation of a new suite of components is within the product management team. It is important that there be significant contributions from the sales and professional services teams. Typical functional representation will be 50% Product Management, 20% Professional Services, 15% Sales, and 15% Engineering.

Requirements Analysis

The second phase is a more detailed analysis of the functional requirements from a standpoint of implementing the functionality of a set of components. Typical functional representation will be 60% Product Management and 40% Engineering.

Architectural Design

The process of turning the functional requirements into an object oriented model that encapsulates the data model and the process model. At this point the weighting shifts to 75% Engineering and 25% Quality Assurance.

Detailed Design, Implementation and Test

This phase consists of a more detailed version of the architectural design, implementation of the business logic, and software testing of the components in hand. At this point the typical functional representation will be 70% Engineering, 20% Quality Assurance, and 10% Documentation.

Deliver Component

The process of delivering the components in a timely manner into the market place. At this point the weighting will be 50% Quality Assurance, 30% Documentation, 20% Support, and 10% Engineering.

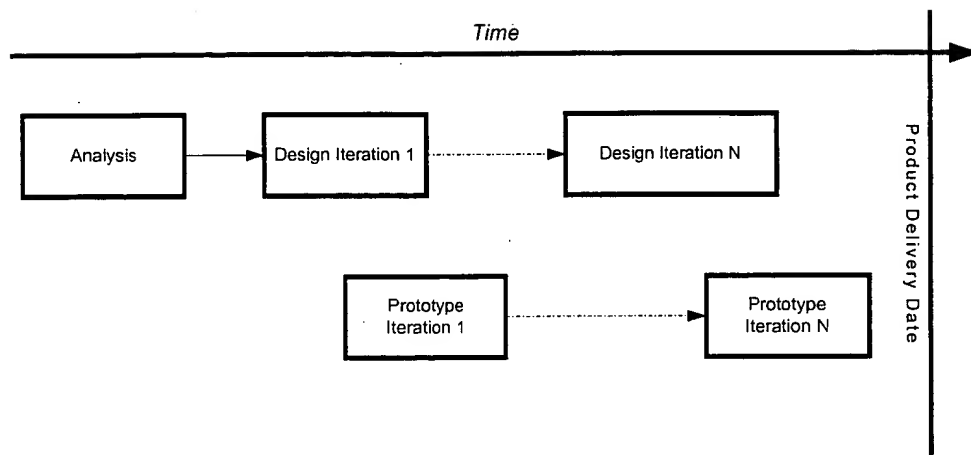
Change Requirements Analysis

The feedback from the customers is analyzed to incorporate the changes in the newer versions of the components. It is important to delegate the required upgrade based upon its type (see Appendix A) to the corresponding phase in the process. The typical weighting at this point is 60% Support, 20% Sales, and 20% Professional Services.

Parallel Development

In many cases there are opportunities to improve time to market by performing steps in parallel. This is especially true with regard to the implementation of the components and the building of the demonstration or prototype application. It is expected that once the initial design of the components is complete, the implementation of the components and the building of the prototype will provide opportunities for improvement.

In general the development process will involve parallel iterations of the product and associated demonstration applications.



Target Functionality

In almost all cases our objective is to establish an immediate product presence in the Enterprise Java Beans Component market. In many cases this will mean replicating a base line product offering in our first iteration. In such cases our advantage will be in the fact that we are EJB and that the components are built as eBSCs and are extensible and configurable. In follow on iterations we will push past the state of the art and deliver functionality that exceeds that of our competitors.

Versioning of Components

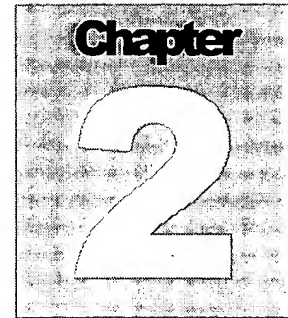
In all cases, our primary concern is to keep our customers on the newest version of our components. The new version might include not only new functionality, but also bug fixes to code they already use, performance improvements, support for new platforms etc. The versioning scheme for our components tries to make upgrades easy, predictable and desirable is described in Appendix A.

Increased Quality

The delivery of high quality software components is an extremely overriding concern in today's market. Use of such reusable software components will result in a system with increased quality since these components are pre-fabricated (i.e., pre-implemented) and pre-tested (i.e., certified) components. The greater the reuse of a component, the greater will be the quality of the system using it since the errors must have been eliminated.

Reduction in Development Costs

Increased quality results in decreased development costs. Use of pre-tested reusable software components decreases the cost incurred in performing various quality assurance activities to detect the errors in the design specifications and implementation.



Analysis

Defining the scope of the business functionality.

The first phase of each New Component Development undertaking defines the scope of the business functionality for a new set of components from the Theory Center. This phase involves a small team including one or two member from the NCD team and additional domain knowledgeable resources from the architecture or professional services teams. The key deliverable from this process is the completion of the NCD Definition.

The NCD Definition is a single document that summarizes research into an entire business domain. The "divide and conquer" approach is taken such that once a wide reaching NCD is completed it is likely that it will be divided into additional sub domains that are then recursively refined as NCD's of their own. This process continues until manageable units of functionality are arrived at.

The NCD Definition itself is composed of the following major sections.

Summary of Scope

The document begins with an overview of the business problem that is being addressed. The purpose of this section is to set the context upon which the rest of the research is to be completed. The content should address the concerns of a typical end user of the targeted solution. The tone should be that of an expert in the field explaining the issues to an outsider.

List of Inputs

Once the scope of the business problem is defined, the process of identifying resources that relate to the problem are identified, assembled and summarized. The following major groupings are specifically addressed.

Interacting Components

This section describes components or other parts of the system that this component will interact with. It should just be a simple bulleted list of items naming the component and what functionality is needed from that component. The list should strive for completeness as it will not only be useful for helping readers of this NCD better understand the component, but it will also help with cross-NCD dependency tracking. In this phase this section might include some components that may or may not actually be interacted with. An example section is:

- Interacts with Inventory component to know what is in stock.
- Interacts with the Shipping component to determine how much shipping will cost.
- Might use the Payment component to authorize payments.

Industry Analysts

In many cases analysts have initiated much of the research into a given sector. Gathering and reading targeted analysts reports is generally the first step towards gaining an understanding of the domain.

Related Industry Standards

Another prime source of information are standards bodies and or professional societies. These organizations represent the contribution of the best thinking in a given domain. They often have the aim of publishing standards that represent the functionality common across an array of vendors. The collected publications and standards are collected into a repository. The list of member companies in such consortiums is a window onto the collection of vendors that are offering solutions in a given domain.

Commercial Packages (Competitors / Partners)

The reference documentation and feature sets of commercial software packages are also gathered. The superset of the most prominent features is the ultimate solution to the problem that is being addressed. The union of the features in these products generally represents the baseline functionality for a product offering. During this process it is important to foster positive relationships with vendors. In many cases we may simply be providing a generic interface that will be used to wrap third party solutions. Such "adapters" are only possible with the cooperation of the partner.

Related Engagements and System Integrators (Customers)

The work that is accomplished in extending our components to meet the needs of our customers is one of our best resources. It is for this reason that it is important to include representatives from the architectural and professional services teams in this part of the process. A corporate knowledge base of documentation from all professional services engagements must be maintained and referenced for recoverable intellectual property.

Additional In-house Resources

Many of the companies best resources lies in the past experience of the current staff. For this reason it is important to publish the goals of the NCD to the rest of the company and elicit input from all interested parties. The presence of such resource should be noted and they should be included as advisors during the second phase of the process.

eFunction Matrix

Upon completion of the list of inputs the creation of a eFunction matrix is initiated. The eFunction matrix is accomplished by first collecting an all encompassing list of features from all of the inputs listed. The features, or eFunctions, must be described in a summary of less than five words. These eFunctions are collected into groups or packages. Longer descriptions are bound to these short descriptions and cross referenced. The eFunctions are listed as the first column of a grid. A template of the resulting table is presented below.

eFunction Matrix: Shopping

| Package | eFunction | Status | Importance | Difficulty | Competitors | C |
|-------------|---|----------------------|-----------------------|-------------------------|-------------|---|
| | | (0 to 10 = complete) | (0 to 10 = must have) | (0 to 10 = wicked hard) | | |
| CRM | Integration with voice, email and web CRM system | | 5 | | | |
| CRM | Provide metrics on the business operations | | 5 | | | |
| Fulfillment | Can pass data to the Order Fulfillment system | | 10 | | | |
| Invoicing | Discount policies can be applied at customer level | 0 | 7 | | | |
| Invoicing | Discount policies can be applied at invoice level | 0 | 7 | | | |
| Invoicing | Misc. charges can be applied to invoice or packing list | 0 | 7 | | | |
| Invoicing | Discounts can be applied to items in cart | | 8 | | | |
| Invoicing | Total of items in cart can be calculated | | 8 | | | |
| Invoicing | Support for terms of sale | 0 | | | | |
| Item | Items can have dimensions, weight | 0 | 7 | | | |
| Item | Items can have a tax code | 0 | 9 | | | |
| Item | Items can be added into shopping cart | | 10 | | | |
| Item | Quantities of items in shopping cart can be updated | | 10 | | | |
| Item | Items can be deleted from the shopping cart | | 10 | | | |
| Order | Multiple delivery methods | 0 | 5 | | | |
| Order | Allow the purchasing of only a portion of the shopping cart | | 5 | | | |
| Order | Sending shipment as a gift | 0 | 9 | | | |
| Order | Customization of order cost calculation policy | 10 | 9 | | | |
| Order | The system works for known users | | 10 | | | |
| Order | The system works for anonymous users | | 10 | | | |
| Order | Special instructions for delivery | 0 | 6 | | | |
| Order | Support for back order cancellation date | 0 | 9 | | | |
| Payments | Can manage multiple payment methods | | 5 | | | |
| Payments | Can manage coupons and gift certificates | 0 | 5 | | | |
| Payments | Can pay with purchase orders | | 6 | | | |
| Payments | Can authorize payment methods | | 7 | | | |
| Payments | Can manage at least 1 payment method | | 8 | | | |
| Payments | Can pay with credit cards | | 10 | | | |
| Shipping | Exact shipping cost can be calculated | | 2 | | | |
| Shipping | Approximate shipping cost can be calculated | | 5 | | | |
| Shipping | Customization of shipping cost calculation policy | 10 | 5 | | | |
| Tax | Tax can be calculated on a line by line basis | 0 | 9 | | | |
| Tax | Interact with ad servers to cross-sell and up-sell | | 4 | | | |
| Tax | Interact with ShoppingAdvisor to recommend products | | 7 | | | |
| Tax | Maintain a personalized shopping list | | 7 | | | |

The table is divided into 5 major sections. The first section lists each "eFunction" and the package or grouping to which it belongs. The second section lists the development status with regard to any pre-existing functionality in house and the importance of the feature as a gauged by competitors, customers, and partners. The Competitors section contains a column for each software solution that can be considered a competitor. For each eFunction an ad-hoc scoring from 1-10 is provided where 10 is a complete solution to that eFunction. The Customers section provides a place to record both customer requests as well as cases in which customers have implemented or extended our components to encompass a function that we did not provide. The Partners section is where opportunities to leverage partnerships are recorded. In many cases it may be difficult to distinguish partners from competitors until the analysis is completed.

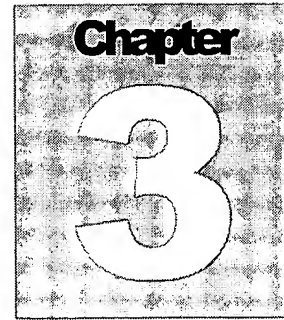
This matrix is important because it provides a single place in which all of the options are quantified and can be compared and contrasted.

Action Items

The final deliverables from the first phase include:

1. A list of targeted features
2. The team composition for the second phase.

Once these deliverables are done, the stage is set for the beginning of the second Phase of the NCD process.



Design

Object Oriented Analysis and Design (OOAD) using UML

The focus of this phase is to perform an in-depth analysis of the resources defined in the first phase. The second phase is used to refine the general descriptions created in the first phase into a design document from which implementation can be started. From the phase one documentation, the actors and then the use cases will be extracted. The use cases can then be used to fill in the detail for each of the components that are to be constructed.

Overview of Usage

A more refined version of the Summary of Scope from the first phase is completed. This defines the specific functionality that will be covered in the first implementation of this NCD. This allows the team to specifically state what will and will not be covered in this release. In addition it is often good practice to describe a future course of action for implementing extended functionality.

Interacting Components

This section describes the refined version of the section, interacting components from the analysis phase. The components or other parts of the system that this component may not interact with will be filtered out in this phase. It will be a list of specific components, whose functionality will be needed by the component in hand.

A Glossary of Domain Specific Terms

It is important to begin by defining the terms that may be ambiguous or unfamiliar to the reader. This will ensure that the content which follows is clearly understood.

A List of the Actors and a Description of Roles

The individual users and or external systems that are to interact with the newly developed component must be listed. This includes:

- End users

- Institutions providing a particular service
- Specific proprietary legacy systems
- Systems accessed by industry standard protocols

Such entities must be identified and their role in the business process clearly explained.

A Collection of Use Cases

Use cases are as defined in the Unified Modeling Language (UML). The UML is an industry standard notation for describing object orient systems. The use cases describe the business process in simple narrative form. The relationships between the actors and the use cases are visualized in use case diagrams and then these use cases are transformed into interaction diagrams that describe the operations that actors initiate on objects as well as object-to-object operations. Their purpose is to help the developer of the system identify the components and the operations that will need to be performed upon them. For more on use cases refer to **The Unified Software Development Process**, by Ivar Jacobson, Grady Booch, and James Rumbaugh. In many cases it is helpful to begin the development of a prototype as part of the use case exercise. This prototype application will often take the form of a graphical front end that implements the system from the actors perspective. This will provide the designers the opportunity to explore the use cases in depth.

Mapping Protocols to Objects

Beyond the standard mechanisms provided in the UML for defining components, it is also useful to consider some other techniques designed to exploit existing message passing formats. In the case where an existing message format exists, such as a Document Type Definition (DTD) for the eXtensible Markup Language (XML), the following guidelines can be used to create an initial component model:

1. Objects should be identified from the message formats. For instance, if there are "New Order" and "Cancel Order" messages in a given protocol it is likely a good idea to create an "Order" object. In many cases the existence of the telltale create, refresh, update, and delete (CRUD) messages signify that there is some underlying entity that needs to be modeled.
 2. The methods on each object should be identified. Many times this will be simply translating a message such as "Cancel Order" to the "cancel" method on the "Order object". Other times this might involve combining multiple messages into a single method call. For example, the messages "Begin Transaction", "End Transaction" and "Update Order" might just be translated into an "update" method on the "Order" object that sends all three messages in the appropriate order.
-

3. The identity of the remote object is captured in a set of attributes that capture the primary key of the entity. Typically the values will be named appropriately as "identifier", "name", or "key".
4. Look also for session based components that simply specify that some action is to be performed but do not identify an entity.

Initial UML Models with Documentation

Once the use case analysis is complete the initial component models will be completed using the UML class diagrams and the Theory Centers specialized stereotypes. The process of applying the Theory Center's mapping of UML to Enterprise Java Beans is described in the document "Modeling with eBusiness Smart Components".

Test Cases

The creation of test cases is the last deliverable in the design phase. The test cases provide an important validation of the design. By designing the test cases before implementation begins, the Quality Assurance team is given an opportunity to have input into the design before implementation has begun. Performing this simple step early on lays the groundwork for having high quality components.

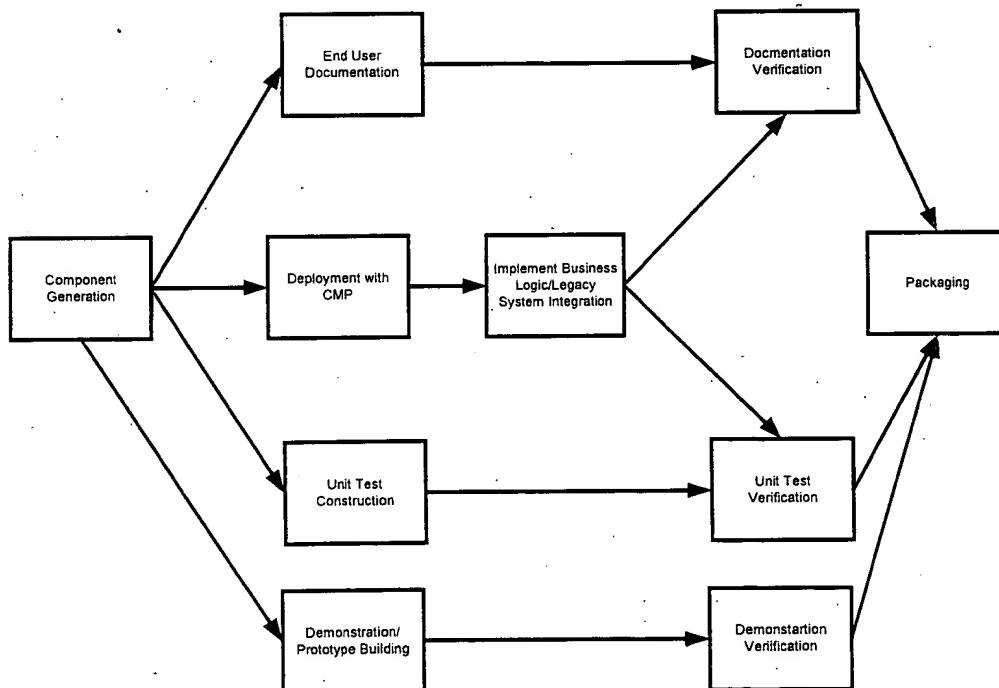
Design Review

The completion of the Design Phase is marked with a formal review. All of the above deliverables are reviewed for content and form. The design review will include senior development staff from Engineering, Architecture, and Professional Services, as well as representatives from the Sales and Marketing teams. This will provide the opportunity for the organization to provide additional feed back before implementation begins and to buy in to the target components

Implementation

Building the New Components.

The third phase of the New Component Development is the implementation of the components themselves. At this point in the development the relational mappings and deployment descriptors are generated. This will include the definition of security roles. A complete set of unit tests that exercise the functionality will be included. This phase may be completed in parallel with implementation of a reference implementation. The implementation will proceed within the following guidelines.



Component Generation

The first step in the creation of implementation is the generation of the classes that represent the components and their Java Doc from the model. The components are then compiled and deployed with the simplest form of persistence. The completion of deployable components makes it possible to begin the implementation, unit test creation and documentation in parallel.

Deployment with Container Managed Persistence (CMP)

This involves installing the bean class and its supporting classes in the EJB server with container-managed persistence. Using this type of persistence, the container must keep the object and the database synchronized i.e., the container is responsible to fetch the data from the database and put it in the bean and also to write the data back to the database (the *ejbload()* and *ejbstore()* methods will not be used to access the database). This is the path of least resistance implementation that allows the creation of a non-optimized solution that can be used in the creation of business logic, unit tests and the reference implementation.

End User Documentation

This involves creating documentation that is targeted at the end-user developer. The documentation team will use the design documentation and the java documentation and prepare it for consumption by outsider parties. The documentation to this point has been focused on determining what the functionality should be. As such it contains references to functionality that we may choose not to implement and product comparisons that must be eliminated.

Unit Test Construction

The QA team will begin the design of tests that fully exercise the components. This process begins before the components have been fully implemented. It is important that the test plans be created and written by a team that is not biased by knowing the implementation details of the components (i.e.: "black-box" testing).

Implement Business Logic/Legacy System Integration

Legacy system integrations is concerned with the integration of legacy applications and data into a distributed computing environment. In many instances the implementation of the business logic will not be object-oriented but must appear to be object-oriented in distributed environment as these applications are of greater use. Software components can be used as a "wrapper" to make these applications appear as though they are object-oriented. Business logic may be added in the form of methods and the implementation of this business logic amounts to functional substance to the components.

Documentation Verification

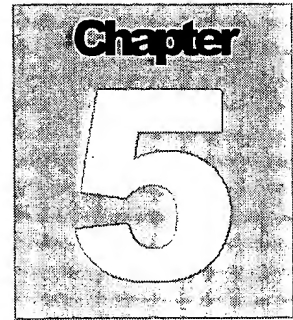
Upon completion of the implementation and the documentation, the two must be brought back into synch. This should focus on confirming that the documentation matches the implementation, as additional details may have been uncovered during implementation that required functionality changes that now need to be documented.

Unit Test Verification

Upon completion of the implementation and the unit tests, the components must be verified.

Packaging

The creation of a reference implementation using the components and the packaging with regard to deployment sets is the final phase of the implementation.

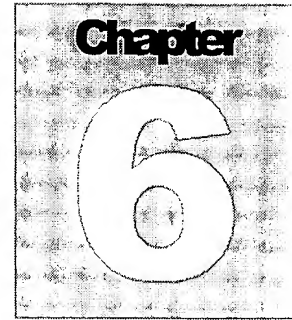


Demonstration

Creating a reference implementation

The final phase of the NCD process involves the creation of a demonstration application that fully utilizes the components in a functioning application. This is a final step that proves out the design and makes the components accessible to the business population. A different team than those who built the components should assemble the application. This will help to ensure that the documentation and design will make sense to our user community. Any required changes to the design or documentation will be made in an update that follows.

In many cases the development of the demonstration application can begin prior to the completion of the implementation. In particular the design of the user interface and the logical flow will be based off of the documentation that results from the design phase. Beginning this process as early as possible will help to discover any flaws in the design before the implementation has proceeded too far.



Release and Support

Releasing the product to customers and supporting it.

Once the product development is complete and a reference implementation has been created, it must be released. This includes the gathering of the documentation, updates to the release notes (as needed), creation of the installable image (e.g.: an InstallShield image), final testing of the installable image, and distribution of the product (e.g.: we b release).

After release customer support must help with installation issues, configuration and usage problems. Also, customer support must funnel back bugs, requests for new features and other comments and suggestions from the customers to the rest of engineering.

Appendix A

Versioning of Components

Introduction

It is in both our customers' and our best interest to keep our customers on the newest version of our components. It is in our customers' best interest because they get the latest and the greatest. This might include not only new functionality, but also bug fixes to code they already use, performance improvements, support for new platforms, etc. It is in our interest because no matter how few versions of our software we want to support, in reality we must support whatever versions our customers are using. Therefore, having them all on the newest version (or at least almost the newest version) will greatly reduce the number of releases we must support.

In any product, managing backwards compatibility such that development progress is not halted and current customers are not angered by the constant changes is a difficult challenge. The challenges become greater as the software product includes more programmability – it is easier to change one's mouse motions to accommodate GUI changes than it is to reprogram all of one's Excel macros because the macro language just changed. Our components, by their very nature, are used as integral portions of large and complex programs making managing backwards compatibility for us an even greater challenge for us than it is for most software vendors.

This situation is complicated by the fact that we are developing new releases at a very rapid rate, currently four per year. This means that if we only support 12 months worth of product we would end up supporting three or four active products while we are in the midst of developing yet another. Such a support effort will surely grind us to a halt. Furthermore, as our customer base continues to grow, we must make upgrading to new versions **easy**, **predictable** and **desirable**. If not we will spend more and more of our time handholding the customer through upgrades, explaining to them what the problems are as they try to upgrade and arm-twisting them to upgrade.

The remainder of this document describes an approach to developing new versions of our components that would try to make upgrades easy, predictable and desirable. The approach taken is based off of the recognition that there is a tradeoff between the complexity of changes we can add to a release and the ease with which the customer can upgrade. That is, the more we restrict the types of changes between releases, the easier an upgrade will be.

Different Types of Upgrades

Following is a list of types of upgrades where each successive type of upgrade gives us more flexibility (i.e.: allows greater enhancements) but also requires more work on the part of the customers when they upgrade.

| Upgrade Type | Types of Changes Allowed | | | Requires Update | | | |
|----------------|--------------------------|--------------|--------------------------|-----------------|--|----------|--------------------------------|
| | Bug Fixes | New Behavior | Modify Existing Behavior | Server | Development Machine | Client | Model and Software |
| Bug Fix Only | Yes | No | No | Required | Only if changes are needed on development machine. | No | No |
| Added Behavior | Yes | Yes | No | Required | Required | Required | Only if new features are used. |
| Major Release | Yes | Yes | Yes | Required | Required | Required | Required |

Bug Fix Only: The bug fix only upgrade is characterized by the fact that it introduces no (let me repeat, "no") changes to the model or to the behavior of any method. It will typically consist of bug fixes only although performance improvements, documentation changes and other modifications are also permissible. Belongings, being both server and client entities, cannot be upgraded at all. The user can install a Bug Fix upgrade without changing any code or doing anything to their Rose models. They just install the product on the server and developer machines and continue working as before. Some releases will not even need to be installed on the developer machines. For example, performance improvements or fixes for bugs that only happen when the component is under heavy load are probably not needed on the development machines.

Added Behavior: In addition to allowing for bug fixes, this upgrade type allows for the addition of new classes as well as for the addition of new methods to existing classes. A new method is either a method of a new name, or a new overload for an existing method. This type of upgrade does not allow for the deletion of an existing method or the changing of behavior of an existing method. Since this type of upgrade allows a class definition to change it will need to be installed on all servers, clients and development machines.

Major Release: This is an all-is-fair release. That is, this release makes no commitment to backward compatibility in any way. It will need to be installed everywhere and existing code will need to be ported to the new components.

Please note that this should not be misconstrued as an exhaustive list of all upgrade types. The three upgrade types discussed here have close cousins, each with slightly differently semantics. For example, we could have a Client Bug Fix upgrade that would be just like a Bug Fix upgrade but it would have to be installed on each client and developer machine as well. This would allow, among other things, changes to Belongings. Another possible upgrade type would be a New Component upgrade that allows bug fixes and the addition of new components but does not allow changes to existing ones. The three upgrade types discussed here are just three reasonably spaced points across the broad spectrum of possible upgrade types.

Naming the Releases

Note: This section is pending information about versioning and naming from BEA. It will be adjusted to satisfy any BEA conventions as needed.

The releases should be named to inform the customer what type of changes they are getting in an upgrade, how major the changes are and how difficult the upgrade will be. The following naming convention meets these requirements:

| | Type of Release | Examples of Release Name |
|----------------|-----------------|--------------------------|
| Bug Fix Only | Dot-dot release | 2.0.1, 2.3.5 |
| Added Behavior | Dot release | 2.1, 2.2 |
| Major Release | Dot-oh release | 2.0, 3.0 |

Technological Assistance

We can use some technological solutions to simplify the implementation of these policies in the following ways:

1. Make the upgrade easy for the customer by simplifying the work they need to do to upgrade existing models in Rose.
2. Ensure that we do not violate a release's constraints. For example, we must ensure that a bug fix release does not add new methods or objects.
3. Simplify the cost of upgrading to the next major release by allowing for parallel development.

Upgrading Rose Models

When a user wants to work with our components in Rose they do so by utilizing our .CAT files (Rose Category files). By doing this, the model for our components are stored in different files than the model for the user's components and classes. By updating our .CAT files, we can cause Rose to add both new components as well as new behavior to existing components without requiring the user to make any change in their model. Once the .CAT files are updated, the next time the user opens a model utilizing our components they will see the upgraded model.

Ensuring a Release's Constraints Are Met

The constraints on an Added Behavior upgrade are that the behavior of existing methods is kept the same and that no methods are deleted. Unfortunately it is impossible to ensure that the behavior of existing methods stays the same (remember the Halting Problem). We can however write a tool to ensure that no methods are deleted. The tool would simply run javap on the old and new class definitions and compare the output (this is trickier than a simple difference but not difficult). A bug fix release has the added constraint that it does not change any class definitions at all. Another tool (or the same tool with a different option) would do a difference and ensure that the class definitions have not changed at all.

Parallel Development

A major release places a large burden on the customer because they must rework their model and the corresponding code. Because of this, we cannot expect them to upgrade an entire large system all at once. There are three scenarios for partial upgrades:

- a) A developer might begin work using a new major release of our components while still maintaining a release using an older release. It would be convenient if they could do this on the same machine.
- b) If they are running multiple applications on their web server, time may force them to upgrade some applications but not others to the newest major release.
- c) They may wish to upgrade some parts of a system to the new components but not the rest. That is, they may wish to have a single application that uses multiple versions of our components.

By changing our package naming scheme to include the Major Release number we can ensure that two releases of our components can reside in the same VM. This will enable scenarios (a) and (b) above. Enabling the partial upgrade described in (c) is significantly harder and while perhaps possible (through the use of the Adapter pattern), will not be further discussed here.

Conclusion

This versioning scheme is far from complete. Investigation still needs to occur to determine exactly what should and should not be allowed in each type of upgrade. For example, should we not bother with Bug Fix upgrades and instead have Client Bug Fix upgrades? The trade-offs need to be better understood before final decisions can be made. Furthermore, this scheme is limited to versioning of our components. A similar scheme will need to be developed for our tools.

That being said, this versioning scheme has two key features, **predictability** and **ease of installation**. It is **predictable** because our customers can tell simply by comparing version numbers what types of changes are in a new release as well as what the complexity of the upgrade will be. By **easily** being able to upgrade to new minor releases (i.e.: dot and dot-dot) we can greatly limit the amount of support we provide. For example, if, in a year we release 2.0, 2.0.1, 2.1, 2.2 and 3.0, we can tell our customers that we only are supporting 2.2 and 3.0. Due to the backwards compatibility constraints of minor releases all 2.x customers using versions prior to 2.2 can easily upgrade to 2.2 without jeopardizing their work (we, of course, will never introduce a regression and will assure our customers of that ☺).